Fundamentos de la Programación
Proyecto 2 – Fase 1: *GiggleBot*

Ingeniería de Robótica SW
GSyC, ETSIT, URJC

Noviembre de 2019

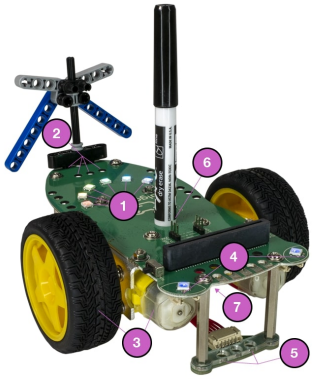
Resumen

GiggleBot es un robot diseñado y fabricado por Dexter Industries, compañía americana que lleva 10 años trabajando en robots para entornos educativos, primero para *Lego Mindstorms* y posteriormente para *Raspberry Pi*. *GiggleBot* es su primer robot diseñado para *micro:bit*.

En esta práctica se presenta el *GiggleBot*: tendrás que aprender a controlarlo desde un programa Python que ejecutará el *micro:bit* que se inserta en el GiggleBot.

1. *GiggleBot*

En la caja que te entregamos encontrarás un *GiggleBot*, incluyendo sus dos ruedas, un sensor laser para medir distancia a objetos y un zócalo para montar el sensor. El robot necesita 3 pilas AA para funcionar de forma autónoma.



El *GiggleBot* contiene:

- En la parte frontal:
  - 2 LEDs RGB y 2 sensores de luminosidad (4)
  - 2 conectores I2C para sensores adicionales (justo debajo de los sensores de luminosidad)
  - 2 sensores para seguir líneas (5)
- En la parte inferior:
  - 2 ruedas motorizadas (3)
  - zócalo para 3 pilas AA (7)
- En la parte superior:
  - zócalo para insertar un *micro:bit*, con el display hacia la parte delantera del *GiggleBot*
  - 7 LEDs RGB en semicírculo (1)
  - 2 conectores para servos (6)
  - un orificio central para colocar un rotulador que permita dibujar al *GiggleBot*
  - varios orificios tipo Lego para completar el *GiggleBot*

2. Entorno de programación

El *GiggleBot* es controlado por el programa que se ejecute en el *micro:bit* insertado en su zócalo.

Puedes utilizar cualquiera de los entornos de programación del *micro:bit*. En esta asignatura utilizaremos para programarlo el lenguaje MicroPython, utilizando el editor `mu-editor`.

3. Instalación del firmware con los módulos del *GiggleBot*

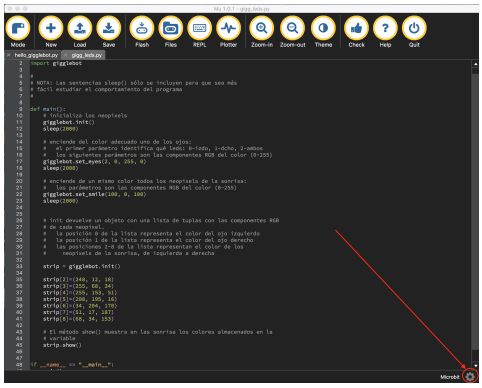
Para programar el *GiggleBot* desde el *micro:bit* se utilizará un conjunto de módulos, entre los que se encuentran:

- `gigglebot` : Módulo principal para interactuar con los componentes básicos del robot
- `distance_sensor` : Módulo para interactuar con el sensor adicional de distancia.
- `gb_diag` : Módulo para acceder a características avanzadas del *GiggleBot* para diagnóstico y depuración.

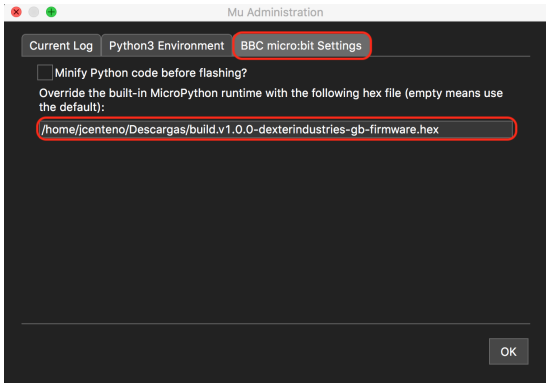
En vez de introducir los ficheros `.py` de los módulos en el micro:bit y que sean cargados desde el programa principal al ejecutarse el `import`, utilizaremos un firmware modificado del *micro:bit* que incluya ya estos módulos sin necesidad de ser cargados desde un fichero, lo que ahorra utilización de memoria del *micro:bit*.

Para instalar el firmware modificado:

1. Descarga el fichero `.hex` con el firmware desde: `https://github.com/DexterInd/micropython-gigglebot/releases/download/v1.0.0/build.v1.0.0-dexterindustries-gb-firmware.hex`
2. En `mu-editor`, pulsa en la esquina inferior derecha el botón de configuración:



3. Elige la pestaña de "BBC micro:bit Settings" y escribe la ruta completa hasta el fichero con el firmware que te has descargado:



En esta URL tienes una guía de programación del *Giglebot* con MicroPython: `https://gigglebot.readthedocs.io`. Explora la documentación para que conozcas las funciones disponibles y su especificación.

4. Ejercicio 1: Programando los neopixels

Tanto los dos neopixels frontales (ojos) como los 7 traseros (sonrisa) son programables para que cada uno muestre un color RGB.

Estudia el siguiente programa de ejemplo (`gigg_leds.py`):

```
from microbit import *
import gigglebot

#
# NOTA: Las sentencias sleep() sólo se incluyen para que sea más
# fácil estudiar el comportamiento del programa
#

def main():
    # inicializa los neopixels
    gigglebot.init()
    sleep(2000)

    # enciende del color adecuado uno de los ojos:
    #   el primer parámetro identifica qué leds: 0-izdo, 1-dcho, 2-ambos
    #   los siguientes parámetros son las componentes RGB del color (0-255)
    gigglebot.set_eyes(2, 0, 255, 0)
    sleep(2000)

    # enciende de un mismo color todos los neopixels de la sonrisa:
    #   los parámetros son las componentes RGB del color (0-255)
    gigglebot.set_smile(100, 0, 100)
    sleep(2000)


    # init devuelve un objeto con una lista de tuplas con las componentes RGB
    # de cada neopixel.
    #   la posición 0 de la lista representa el color del ojo izquierdo
    #   la posición 1 de la lista representa el color del ojo derecho
    #   las posiciones 2-8 de la lista representan el color de los
    #     neopixels de la sonrisa, de izquierda a derecha

    strip = gigglebot.init()

    strip[2]=(248, 12, 18)
    strip[3]=(255, 68, 34)
    strip[4]=(255, 153, 51)
    strip[5]=(208, 195, 16)
    strip[6]=(34, 204, 170)
    strip[7]=(51, 17, 187)
    strip[8]=(68, 34, 153)

    # El método show() muestra en las sonrisa los colores almacenados en la
    # variable
    strip.show()


if __name__ == "__main__":
    main()
```

`init()` inicializa todos los neopixels, dejando la sonrisa apagada y los ojos con color azul (si se usan las baterías y están suficientemente cargadas) o rojos (si se usan las baterías y no están suficientemente cargadas, o si no se usan las baterías por estar el *micro:bit* conectado a un puerto USB). No pueden usarse los neopixels sin llamar previamente a `init()`. Si no vas a usar los neopixels en un programa, es mejor no llamar a init() para conservar memoria.

Ejercicio: Realiza un programa que muestre en la sonrisa del *gigglebot* los colores rotando entre todas las posiciones:

Sonrisa rotando colores

Haz click encima de esta figura para que comience la animación.

Figura 1: OJO: evince tiene algunos problemas con la reproducción. Utiliza la aplicación okular para visualizar este documento PDF. Si no lo tienes instalado, utiliza el comando `sudo apt install okular` para instalar okular en Ubuntu 18.04

5. Ejercicio 2: Moviendo el *GiggleBot*

Puedes controlar los dos motores del *GiggleBot* para que el robot avance, retroceda o gire.

Estudia el siguiente programa de ejemplo (`gigg_mov.py`):

```
from microbit import *
import gigglebot

def main():

    # Fija la velocidad de los motores izquierdo, derecho, de -100 a 100.
    # Si la velocidad es negativa, el motor gira hacia atrás.
    # Si la velocidad de los dos motores es la misma, el robot debería
    # avanzar hacia adelante, si no lo hiciera, ajusta levemente la
    # velocidad de uno de los dos motores
    gigglebot.set_speed(50, 50)

    # Mueve los dos motores a la velocidad fijada, y
    # hacia adelante si la velocidad fijada es positiva
    # El segundo argumento es el número de milisegundos, transcurridos los
    # cuales los motores se paran.
    gigglebot.drive(gigglebot.FORWARD, 1000)

    # Mueve los dos motores a la velocidad fijada, y
    # hacia atrás si la velocidad fijada es positiva
    # El segundo argumento es el número de milisegundos, transcurridos los
    # cuales los motores se paran
    gigglebot.drive(gigglebot.BACKWARD, 1000)

    sleep(1000)

    # Gira el robot hacia la izquierda si la velocidad fijada es positiva,
    # girando uno de los dos motores
    # El segundo argumento es el número de milisegundos, transcurridos los
    # cuales los motores se paran
    gigglebot.turn(gigglebot.LEFT,500)

    # Gira el robot hacia la derecha si la velocidad fijada es positiva,
    # girando uno de los dos motores
    # El segundo argumento es el número de milisegundos, transcurridos los
    # cuales los motores se paran
    gigglebot.turn(gigglebot.RIGHT,500)

    # Para el robot (innecesario en este ejemplo)
    gigglebot.stop()

    # NOTA: Sin especificar en estas funciones un tiempo, el movimiento
    # será indefinido, hasta que se llame a otra de las funciones o a stop()

if __name__ == "__main__":
    main()
```

Ejercicio: Coloca el *gigglebot* sobre varias hojas de papel. Inserta un rotulador de tamaño adecuado en el orificio central del robot. Haz un programa que consiga que el robot dibuje en el suelo un círculo lo más perfecto posible de unos 40 cm de radio. Idea: Un enfoque puede ser aproximar un círculo por un polígono de muchos lados.

6. Ejercicio 3: Siguiendo la luz

El *GiggleBot* tiene dos sensores de luz justo delante de los neopixels de los ojos, para detectar la luz ambiente.

Estudia el siguiente programa de ejemplo (`gigg_light.py`) y utilízalo para calibrar los valores qué detectan como más o menos luminosidad los sensores de luz (observa que el programa muestra los valores a través de la interfaz REPL y del trazador del `mu-editor`):

```
from microbit import *
import gigglebot

def main():
    finish = False
    while not finish:
        # Obtiene la lectura de los sensores de luz
        # El segundo parámetro puede ser LEFT, RIGHT o BOTH
        # Si se lee un sólo sensor, devuelve un entero
        # Si se leen los dos sensores, se devuelve una tupla de 2 enteros (izquierdo, derecho)
        # Los valores devueltos están entre 0 y 1024, cuanto mayor, más luz detectada.
        values = gigglebot.read_sensor(gigglebot.LIGHT_SENSOR, gigglebot.BOTH)
        print(values)
        sleep(1000)

        # Si se pulsa algún botón del micro:bit, se sale de bucle y termina el programa
        if button_a.was_pressed() or button_a.was_pressed():
            finish=True

if __name__ == "__main__":
    main()
```

Ejercicio: Haz un programa que haga que el *gigglebot* huya de la luz brillante. Para probarlo, coloca el robot en un entorno de poca luz y mueve delante de él una luz (por ejemplo la del flash de un teléfono móvil), para que el robot huya de la luz alejándose de ella. Idea: toma la luz de ambos sensores, y haz que el robot se mueva hacia el lado donde la lectura detecta luz menos brillante.

Igual que en el ejemplo, haz que el programa, y por tanto el robot, pare al pulsar un botón del *micro:bit*.

Ejercicio: Modifica el programa anterior para que el *gigglebot* se comporte al revés: siguiendo la luz en vez de huyendo de ella.

7. Ejercicio 4: Siguiendo una línea

El *GiggleBot* tiene dos sensores de línea en la parte inferior delantera, casi tocando el suelo. Estos sensores se utilizan de forma muy similar a los sensores de luz.

Estudia el siguiente programa de ejemplo (`gigg_line.py`) y utilízalo para calibrar los valores de superficies que detectan como claras u oscuras los sensores de línea (observa que el programa muestra los valores a través del display del *micro:bit* cuando se pulsa el botón A):

```
from microbit import *
import gigglebot

def main():
    while True:
        if button_a.was_pressed():
            # Obtiene la lectura de los sensores de luz
            # El segundo parámetro puede ser LEFT, RIGHT o BOTH
            # Si se lee un sólo sensor, devuelve un entero
            # Si se leen los dos sensores, se devuelve una tupla de 2 enteros (izquierdo, derecho)
            # Los valores devueltos están entre 0 y 1024, cuanto mayor, más clara es la superficie
            # sobre la que se encuentra el robot.
            values = gigglebot.read_sensor(gigglebot.LINE_SENSOR, gigglebot.BOTH)
            display.scroll(values[0])
            display.scroll(values[1])

if __name__ == "__main__":
    main()
```

Observa este nuevo programa de ejemplo que sigue a continuación (`gigg_line_2`). En él, se compara la lectura de cada sensor con un cierto umbral, por debajo del cual se considerará que se ha detectado una línea negra por el sensor. Cuando el sensor izquierdo detecta el color negro, se enciende el neopixel más a la izquierda de la sonrisa, cuando el sensor derecho detecta el color negro se enciende el neopixel más a la derecha de la sonrisa. Prueba el programa colocando una superficie negra debajo de los sensores, y en su caso, retoca la constante `THRESHOLD` del umbral de detección de línea.

Nota: Puedes obtener páginas con líneas dibujadas imprimiendo algunas de las hojas que puedes encontrar en: `https://gigglebot.io/media/free_downloads/gigglebot_line_follower_2.pdf`

```
from microbit import *
import gigglebot

# umbral de detección de línea
THRESHOLD = 100

def main():
    strip = gigglebot.init()
    while True:
        strip[2] = (0, 0, 0)
        strip[8] = (0, 0, 0)
        values = gigglebot.read_sensor(gigglebot.LINE_SENSOR, gigglebot.BOTH)
        if values[0] < THRESHOLD:
            strip[2] = (255, 0, 0)
        if values[1] < THRESHOLD:
            strip[8] = (255, 0 ,0)
        strip.show()
        sleep(200)

if __name__ == "__main__":
    main()
```

Ejercicio: Haz un programa que consiga que el *GiggleBot* avance por una línea con curvas. Idea: Si los dos sensores detectan la línea, haz que el robot avance, si sólo uno de los dos la detecta, haz que el robot gire un poco hacia ese lado.

8. Ejercicio 5: Midiendo distancias

El sensor laser no viene integrado en el robot. Sirve para detectar a qué distancia está un objeto y se conecta a uno de los dos conectores I2C que tiene *GiggleBot* situados justo por debajo de los ojos.

Este sensor se programa de forma un poco diferente de los dos anteriores.

Estudia el siguiente programa de ejemplo:

```
from microbit import *
import gigglebot
import distance_sensor

def main():

    # Es necesario crear un objeto de la clase DistanceSensor para
    # manejar el sensor de distancia
    ds = distance_sensor.DistanceSensor()

    while True:
        if button_a.was_pressed():
            # Obtiene la lectura del sensor de distancia
            # El valor devueltos es la distancia en milímetros, detecta hasta un máximo de 2300mm
            distance = ds.read_range_single()
            display.scroll(distance)


if __name__ == "__main__":
    main()
```

IMPORTANTE: Tras reiniciar un programa que usa el sensor de distancias es necesario interrumpir la alimentación del *micro:bit*, bien desconectando el cable USB, o bien (si ya está funcionando desconectado) apagando y volviendo a encender el *GiggleBot*. Lee el Apéndice para más detalles.

Ejercicio: Haz un programa que te permita guiar el *GiggleBot* con una mano por delante de él, de forma que se vaya acercando a la mano según la vas alejando, pero se pare si está a menos de 10cm de tu mano. Idea: El robot debe avanzar para disminuir la distancia entre medidas consecutivas.

9. Ejercicio 6: Emitiendo y recibiendo por radio

El *micro:bit* tiene una antena Bluetooth que le permite emitir y recibir. Aunque no es parte del *GiggleBot*, resulta interesante utilizar la radio del *micro:bit* en combinación con la capacidad de movimiento y detección del *GiggleBot*, por ejemplo para que varios GiggleBot puedan comunicarse entre sí.

Puedes consultar la documentación del módulo de radio del *micro:bit* en: `https://microbit-micropython.readthedocs.io/en/latest/radio.html`

Los dos siguientes programas muestran el código que usa un *micro:bit* que envía mensajes (`radio-sender.py`), y el código que usa un *micro:bit* para recibirlos (`radio-sender.py`).

Ponte de acuerdo con alguien para programar el emisor y el receptor en dos *micro:bit* diferentes. Podréis así comprobar cómo se envían y reciben los mensajes, y cómo varían los valores de la potencia de la señal de radio recibida en función de la distancia a la que se encuentran vuestros *micro:bit*. Luego intercambiad los roles para que podáis entender mejor cómo se comporta este código.

Código del *micro:bit* emisor

```
import radio
from microbit import sleep

def main():
    # String con un identificador del emisor, cámbialo para no coincidir con el
    # de otros microbits
    IDENTIFIER = "MICROBIT_10310"

    # Periodo de emisión de mensajes, en milisegundos
    PERIOD = 2000

    # Enciende la radio
    radio.on()

    # Estable la potencia de la emisión, de 0 a 7 con 7 como máximo
    # Mira la documentación para ver otros parámetros que pueden
    # fijarse en radio.config()
    radio.config(power=7)

    while True:
        sleep(PERIOD)
        radio.send(IDENTIFIER)

if __name__ == "__main__":
    main()
```

Código del *micro:bit* receptor

```
import radio
from microbit import display, sleep

def main():
    radio.on()

    while True:
        sleep(1000)

        # Intenta recoger un mensaje que haya llegado al microbit
        # Si no ha llegado ningún mensaje, devuelve None
        # Si ha llegado un mensaje, devuelve una tupla de 3 valores:
        #   una lista de bytes
        #   la potencia de señal recibida, en dBm
        #   una marca de tiempo
        received = radio.receive_full()

        if received != None:
            # no necesitamos la marca de tiempo, por eso la almacenamos en la variable _
            msg, dBm, _ = received

            # Los primeros 3 bytes de msg son una cabecera que descartamos
            # La siguiente línea convierte el resto de bytes a un string utf8
            identifier = str(msg[3:], 'utf8')

            # Muestra en el display el identificador enviado y la potencia recibida.
            display.scroll(identifier + "#" + str(dBm))


if __name__ == "__main__":
    main()
```

Ejercicio: Escribe un programa que pueda funcionar como *GiggleBot* Maestro o como Seguidor. Un GiggleBot en modo Maestro envía por radio instrucciones de movimiento que tienen que seguir los robots que estén en modo Seguidor.

Al arrancar el programa el robot no hará nada hasta que se pulse el botón A o el B:

- Si se pulsa el botón A, escribe en el display del *micro:bit* una "M", enciende todos los neopixels de color rojo, y pone al GiggleBot en modo Maestro. En este modo, el *micro:bit* enviará alternativamente (en un orden preestablecido, elegido por ti) 6 órdenes de entre estas 4:

  - `"FORWARD"`: Orden de avanzar durante 1 segundo
  - `"RIGHT"`: Orden de girar a la derecha durante 0.5 segundos
  - `"LEFT"`: Orden de girar a la izquierda durante 0.5 segundos
  - `"STOP"`: Orden para dejar de moverse

  Estas órdenes serán enviadas por radio a un *GiggleBot* en modo Seguidor. El *GiggleBot* en modo Maestro, a la vez que envía estas órdenes, realiza también los mismos movimientos, por lo que Maestro y Seguidor deberán moverse más o menos de forma similar y simultánea.

- Si se pulsa el botón B, escribe en el display del *micro:bit* una "S", enciende todos los neopixels de color verde, y pone al GiggleBot en modo Seguidor. En este modo, el *micro:bit* esperará a recibir por radio 6 órdenes de entre las 4 distintas que envía el robot en modo Maestro.

  El *GiggleBot* seguidor, al recibir estas órdenes, realizará esos movimientos. Maestro y Seguidor deberán moverse más o menos de forma similar y simultánea. Si hay varios robots seguidores todos ellos recibirán el mensaje y realizarán las mismas acciones de movimiento recibidas del robot en modo Maestro.

Para evitar interacciones no deseadas con otros *micro:bit* cercanos que puedan recibir los mensajes enviados por tu Maestro, puedes probar a cambiar de canal de emisión en `radio.config()`, y a enviar un identificador específico concatenado por delante de las órdenes de tu Maestro. De este modo sólo los Seguidores que estén en el mismo canal y que utilicen el mismo identificador deberán hacer caso a las instrucciones enviadas por tu robot Maestro.

10. Apéndice: Respuesta a preguntas frecuentes

1. **¿Qué hago si tras transferir con FLASH un programa desde `mu-editor` aparece un error de acceso al *micro:bit*?**

   Cierra el editor, expulsa el *micro:bit*, desconecta el cable USB.

   Vuelve a conectar el cable USB, espera unos segundos, y arranca el editor.

2. **¿Qué hago si al empezar el programa en el *micro:bit* aparece un error señalando que no existe el módulo `gigglebot`?**

   Configura `mu-editor` para que transfiera siempre el firmware del *GiggleBot*, como se explica en el apartado 3.

3. **¿Funciona *GiggleBot* mientras el *micro:bit* está conectado al ordenador por el cable USB?**

   Mientras el *micro:bit* está conectado al ordenador por el cable USB, el *micro:bit* recibe la alimentación por el puerto USB. El *GiggleBot* también recibe alimentación por el puerto USB, que permite acceder a los neopixels y a los sensores, **pero no permite operar los motores**. Para que funcionen los motores es necesario que el interruptor del *GiggleBot* esté en ON. Sin embargo, no es conveniente tener en ON el *GiggleBot* mientras se transfiere por FLASH un programa al *micro:bit*, pues puede producirse un encendido imprevisto de los motores.

4. **¿Qué hago si al inicializar el sensor de distancia aparece un `OSError` en los leds del micro:bit?**

   Un fallo en la implementación de la interfaz I2C del *micro:bit* hace que el sensor de distancia necesite desconectarse de la alimentación y volverse a conectar para volver a funcionar correctamente tras haber sido utilizado. Por lo tanto, un programa que utilice el sensor de distancia funcionará sólo la primera vez, y al reiniciarlo con el botón de Reset o con CTRL-D en REPL, aparecerá un `OSError` al llamar al constructor.

   Para volver a ejecutar el programa es necesario interrumpir la alimentación del *micro:bit*, cosa que puede hacerse de dos formas:

   - si el *micro:bit* está conectado al ordenador por el cable USB, desconecta el cable USB del ordenador y vuélvelo a conectar.
   - si el *micro:bit* está desconectado del ordenador, y funciona alimentado por la baterías del *GiggleBot*, apaga y enciende el interruptor del *GiggleBot*.